Adaptation consistante d'ontologies à l'aide des grammaires de graphes

Mariem Mahfoudh[1], Laurent Thiry[1], Germain Forestier[1], Michel Hassenforder[1]

MIPS, Université de Haute Alsace
12 rue des frères Lumières, 68093 Mulhouse Cedex, France
{mariem.mahfoudh, laurent.thiry, germain.forestier, michel.hassenforder}@uha.fr

Résumé : Les ontologies tendent à intégrer le cœur de tout système d'information. Les domaines évoluant sans cesse, les ontologies doivent elles même pouvoir s'adapter. Dans ce contexte, l'article propose la formalisation du concept d'adaptation basée sur les grammaires de graphes, ce qui permet notamment de gérer les changements des ontologies et de définir une approche à priori de résolution des incohérences susceptibles d'être générées. Comme application, l'article considère l'ontologie EventCCAlps développée dans le cadre du projet européen CCAlps.

Mots-clés : ontologies, grammaires de graphes, adaptation, changements ontologiques, règles de réécriture de graphes.

1 Introduction

Définies comme "une spécification explicite d'une conceptualisation d'un domaine" (Gruber, 1993), les ontologies permettent la représentation des connaissances au moyen de classes reliées par des relations et associées à des propriétés. Elles sont souvent utilisées pour la méta-modélisation des domaines évolutifs et ont besoin à cet égard de s'adapter aux changements pouvant apparaitre. Cette adaptation est traduite par une modification de leurs composantes et elle est appelée *changement ontologique*. Les changements sont généralement classés en deux types (Klein, 2004) avec : 1) les *changements élémentaires* représentant une opération primitive qui affecte une seule entité de l'ontologie (ex. ajout de classe) et 2) les *changements composés* exprimant un enchainement de plusieurs changements

élémentaires (ex. fusion de classes). Une mauvaise application d'un changement peut altérer la consistance des ontologies en affectant leur structure et/ou leur sémantique. Ceci promeut donc la nécessité de formaliser le processus d'évolution et de le traiter d'une manière méthodique. Notre travail propose ainsi l'utilisation des grammaires de graphes basées sur les approches algébriques pour exprimer et suivre les changements ontologiques. Les grammaires de graphes présentent un cadre formel rigoureux permettant de vérifier la faisabilité des changements. Elles évitent (grâce à leurs conditions d'application) toute exécution d'un changement qui ne satisfait pas les contraintes définies et permettent ainsi de définir une approche à priori de résolution des incohérences. Elles offrent aussi différents outils avec en particulier AGG (Algebraic Graph Grammar) qui est considéré comme l'un des outils d'usage général (Ermel. *et al.*, 1999).

L'article est organisé comme suit : la section 2 présente les grammaires de graphes. La section 3 propose la formalisation des changements ontologiques avec les grammaires de graphes. La section 4 présente une application dans le cadre du projet européen CCAlps. La section 5 présente certains travaux liés à ce travail. Enfin, une conclusion synthétise le travail présenté et donne les perspectives envisagées.

2 Les grammaires de graphes

Définition 1 (Grammaires de graphes). Une grammaire de graphes (GG) est une paire composée d'un graphe initial (G) appelé graphe hôte et d'un ensemble de règles de production appelées aussi règles de réécriture (ou bien transformation) de graphe.
Une règle est définie par une paire de graphes :

- LHS (Left Hand Side) est un graphe de côté gauche présentant la pré-condition de la règle et doit être un sous graphe de G.
- RHS (Right Hand Side) est un graphe de côté droit présentant la post-condition de la règle et doit remplacer LHS dans G.

Les grammaires de graphes peuvent être typées. Une grammaire de graphe typée est définie par $TGG = (G_T, GG)$ avec $G_T = (N_T, E_T)$, un graphe de type précisant les types des nœuds (N) et des arêtes (E) du graphe hôte G. Le typage de G est donné ainsi par un mapping ou morphisme $t : G \rightarrow G_T$ avec $t : E \rightarrow E_T$ et $t : N \rightarrow N_T$.

La transformation de graphe consiste à définir comment un graphe G peut être transformé en un nouveau graphe G'. Pour cela, il doit exister un morphisme qui remplace LHS par RHS pour obtenir G'. Différentes

approches ont été proposées pour appliquer ce remplacement (Rozenberg, 1999). Les approches algébriques s'appuient sur la théorie des catégories (Ehrig *et al.*, 1973) avec notamment le concept de *pushout*.

Définition 2 (Théorie des catégories). Une catégorie est une structure composée de : 1) une collection d'objets O ; 2) un ensemble de morphismes M et une fonction $s : M \rightarrow O \times O$, $s(f) = (A, B)$ est notée alors $f : A \rightarrow B$; 3) une loi de composition $(\circ) : M \times M \rightarrow M$; 4) un morphisme identité pour chaque objet $id : O \rightarrow O$. La loi de composition doit être associative et avoir l'identité comme élément neutre.

Définition 3 (Pushout). Soient trois objets A, B et C et deux morphismes $f : A \rightarrow B$ et $g : A \rightarrow C$. Le pushout de B et C est une structure (D, i_1, i_2) avec un objet D et deux morphismes $i_1 : B \rightarrow D$ et $i_2 : C \rightarrow D$ telle que $i_1 \circ f = i_2 \circ g$.

A partir de là, deux variantes sont proposées pour la réécriture de graphe : le *Simple pushout SPO* (Löwe, 1993) et le *Double poushout DPO* (Ehrig, 1979). Dans le cadre de ce travail, seule l'approche SPO a été considérée car elle est plus générale (ex. absence de gluing condition) et suffit pour représenter les différents changements ontologiques. A partir de là, appliquer une règle de réécriture (r) à un graphe initial G, selon la méthode SPO, revient à : 1) trouver le LHS dans G ç.à.d chercher un morphisme m : $LHS \rightarrow G$; 2) supprimer de G : $LHS - (LHS \cap RHS)$; 3) ajouter à G : $RHS - (LHS \cap RHS)$. Cette opération se fait par le calcul de pushout et donne une nouvelle version G' de G (Figure 1).

Ainsi, une ontologie sera formalisée par les grammaires de graphes avec G_T représentant le méta-modèle de l'ontologie et le graphe hôte G = (N, E) définissant la base de connaissance avec N représente les classes, les individus, etc et E représente les axiomes.

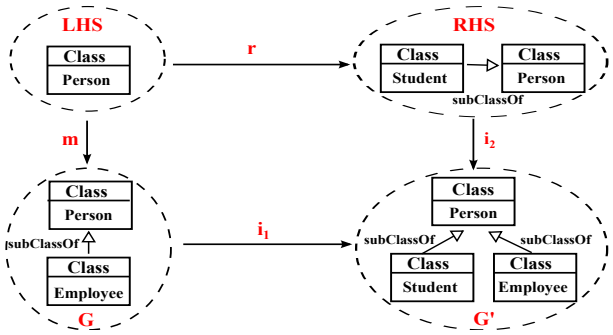

FIGURE 1 – Application d'une règle de réécriture de graphe avec SPO.

3 Formalisation des changements ontologiques

Cette partie propose d'introduire la formalisation des changements ontologiques à l'aide des grammaires de graphes typés. La première étape consiste ainsi à créer le graphe type quant à la deuxième, elle définit les changements ontologiques sous la forme de règles de réécriture de graphes.

3.1 Le graphe de type

Dans le cadre de cet article, c'est le langage OWL [1] qui a été retenu pour décrire les ontologies dans la mesure où c'est le standard du W3C.
La figure 2 présente son méta-modèle représenté en AGG (G_T). Il définit les classes, les propriétés et les individus sous la forme de nœuds attribués avec l'attribut `name` représentant le nom local de l'entité et l'attribut `iri` son identificateur de ressource internationalisé. G_T définit aussi les restrictions de valeur (`HasValue`, `AllValuesFrom`, `SomeValuesFrom`) et de cardinalité (`CardinalityRestriction`). Les axiomes sont représentés sous la forme d'arêtes ou bien des graphes exprimant les relations entre classes, propriétés et individus (ex. l'arête `disjointWith` représente la disjonction entre deux classes ou entre deux propriétés).

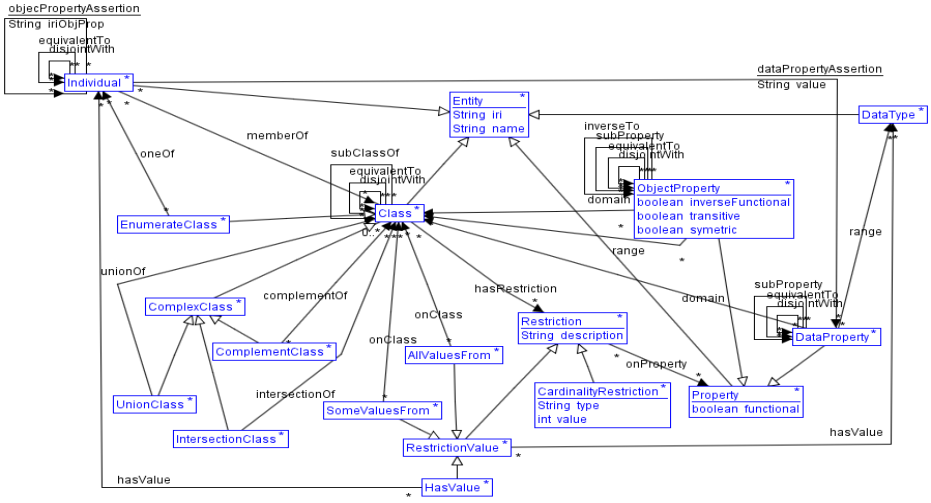

FIGURE 2 – Graphe de type utilisé pour les règles de réécriture.

1. www.w3.org/2004/OWL

3.2 Formalisation des changements ontologiques

Dans ce qui suit, une formalisation de certains changements élémentaires (ajout, suppression, modification) est présentée.

Définition 4 (Changement ontologique). Un changement ontologique est formalisé par un 5-uplet CH = (Nom, NAC, LHS, RHS, CHD) avec : 1) Nom précise le type du changement ; 2) NAC définit la condition qui ne doit pas être vraie pour que la règle de réécriture puisse être appliquée ; 3) LHS présente les pré-conditions de la règle ; 4) RHS définit les post-conditions de la règle ; 5) CHD présente les changements dérivés, des opérations additionnelles devant être jointes à CH afin de corriger les incohérences pouvant survenir.

Ainsi, les incohérences traitées sont essentiellement : 1) Redondance de données, peut être générée suite à une opération d'ajout ou de renommage et elle est corrigée par les NACs ; 2) Nœuds isolés, un nœud N_x est dit isolé ssi $\forall N_i \in N$, $\nexists E_i \in E$ tel que $E_i = (N_x, N_i)$. Cette incohérence nécessite de rattacher N_x au reste du graphe et selon son type, des changements dérivés sont proposés ; 3) Individus orphelins, incohérence générée suite à une suppression des classes définissant des individus ; 4) Axiomes contradictoires, il ne faut pas accepter l'ajout d'un axiome qui contredite un axiome déjà défini dans l'ontologie.

De là, le changement *RenameObjectProperty(OBIRI, OBIRINew)* consiste à renommer un nœud de type `ObjectProperty` (OB). Le LHS est constitué d'un noeud `OB` dont l'attribut `iri` est égal à `OBIRI`. Il sera remplacé par le graphe RHS formé d'un noeud dont l'`iri` est égal à `OBIRINew`. Le NAC devrait être égal à RHS pour empêcher la redondance (Figure 3).

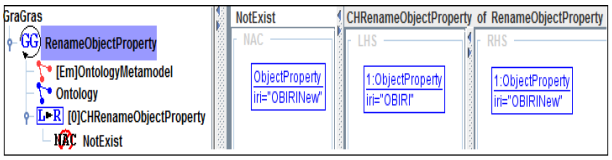

FIGURE 3 – Règle de réécriture du changement RenameObjectProperty.

Le changement *AddClass(Cnew)* permet d'ajouter au graphe G un nouveau nœud de type `Class`. Avec les grammaires de graphes cela revient à préciser que $RHS = Cnew$. Puis, pour empêcher la redondance de données, il suffit d'indiquer que $NAC = Cnew$. Cependant, un nœud

ne doit pas être isolé. Pour rattacher un nœud de type `Class` au reste du graphe, deux types de correction peuvent être appliqués : `AddAxiom` ou bien `AddObjectProperty`. La première consiste à ajouter un nouvel axiome reliant *Cnew* à une propriété déjà existante (en appliquant les changements `addRange` ou bien `addDomain`), ou encore le relier à un autre nœud de type `Class` en appliquant les changements `AddSubClass`, `AddEquivalentClass`, etc. La deuxième alternative de correction consiste à ajouter au graphe une nouvelle propriété dont le nœud *Cnew* est l'un de ses deux membres. La Figure 4, montre la règle de réécriture de changement `AddClass` suivie de certains changements dérivés qui sont classés par couches afin de définir leur ordre d'application.

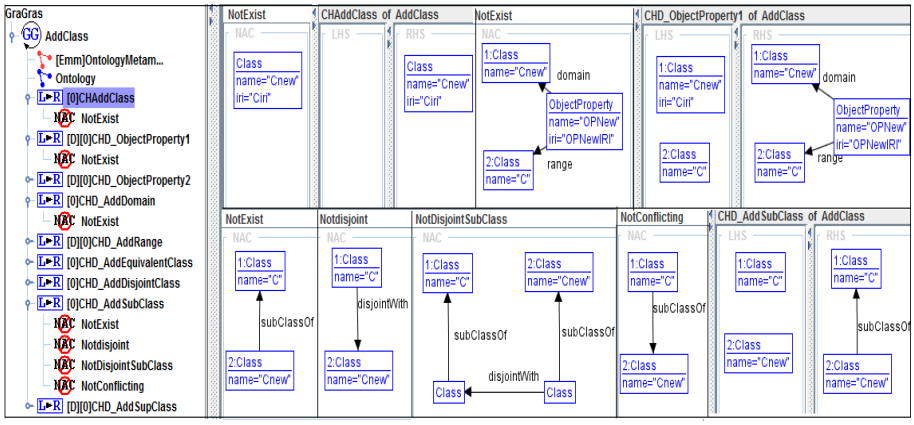

FIGURE 4 – Règles de réécriture du changement AddClass.

Le changement *AddDisjointClass(C1, C2)* consiste à ajouter un lien de disjonction entre deux nœuds de type `Class`. Ainsi, trois NACs sont définis pour vérifier l'absence des arêtes de type : 1) `disjointWith` pour éviter la redondance ; 2) `equivalentTo`, deux classes ne peuvent pas être à la fois équivalentes et disjointes ; 3) `subClassOf`, deux classes partagent une relation de subsomption ne peuvent pas être disjointes.

Le changement *RemoveClass(C)* peut engendrer certaines incohérences comme par exemple, l'existence des individus orphelins ou bien encore le manque des membres d'une contrainte de restriction. A partir de là, avant de supprimer un nœud, il faut vérifier toutes ses dépendances pour en proposer des corrections. En effet, les restrictions devront être supprimées mais le traitement des individus passe par différentes étapes (Figure 5). Ainsi, avant de supprimer une classe (C) possédant des individus *I*, il faut

vérifier : 1) si $C\ subClassOf\ C_p \wedge \forall C_i\ subClassOf\ C_p \wedge !disjointWith\ C$, alors, $I\ memberOf\ C_p$; 2) sinon si $\exists C_i \in G$ tel que $C_i\ equivalentTo\ C$, alors, $I\ memberOf\ C_i$; 3) sinon si $\exists I_i \in G$ tel que $I_i\ memberOf\ C_i \wedge I_i\ equivalentTo\ I$, alors, $I\ memberOf\ C_i$; 4) si aucun de ces cas n'est satisfait les individus orphelins seront supprimés.

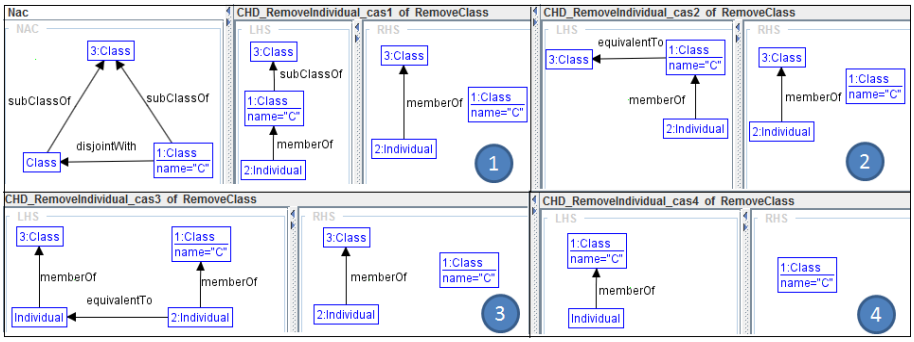

FIGURE 5 – Changement dérivé RemoveIndividual du changement RemoveClass

4 Application

Le travail présenté s'inscrit dans le cadre du projet européen CCAlps [2]. Nous utilisons ainsi l'ontologie OWL EventCCAlps pour illustrer la suppression d'une classe. Cette ontologie définit les concepts liés à des événements. Elle présente leurs caractéristiques (description, période,...) et leurs différentes relations avec les autres concepts (Company, Hub, ...). Pour pouvoir appliquer les règles de réécriture, l'ontologie doit être exprimée en AGG. Pour cela, deux programmes *OWLToGraph* et *GraphToOWL*, basés sur l'API d'AGG et la bibliothèque Jena [3], ont été développés afin d'automatiser la transformation d'OWL à AGG et inversement.
La Figure 6 montre un extrait du résultat de la transformation d'EventCCalps (les IRIs ont été supprimés de la figure pour des raisons de lisibilité).

La figure 7 présente la définition du changement `RemoveClass` (Employee). Comme déjà évoqué, la suppression de classe définit plusieurs cas en fonction des liens de dépendance du nœud. Dans le présent cas, la

2. www.ccalps.eu, numéro de projet 15-3-1-IT
3. jena.sourceforge.net

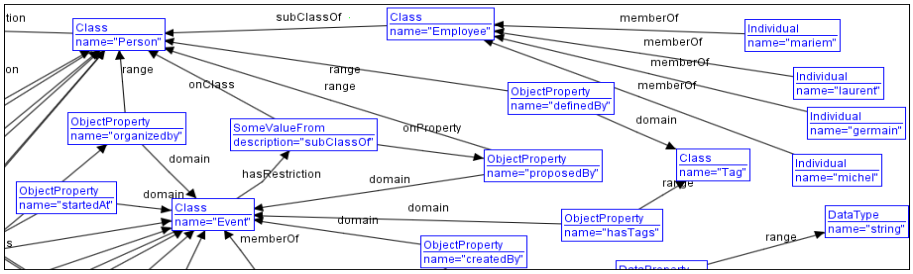

FIGURE 6 – Extrait de l'ontologie EventCCAlps représentée en AGG.

classe `Employee` possède des individus (`mariem`, `laurent`, `germain` et `michel`). Elle est sous-classe de la classe `Person` et aucune classe de l'ontologie n'est à la fois sous-classe de `Person` et disjointe de la classe `Employee`. Ainsi, `RemoveClass` invoque le changement dérivé `RemoveIndividual` et attache les individus de la classe `Employee` à la classe `Person`. De cette manière, les individus et les connaissances peuvent être gardés sans affecter la consistance de l'ontologie.

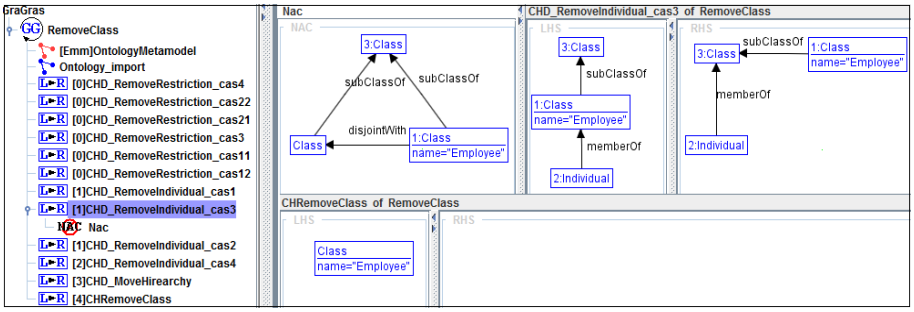

FIGURE 7 – Règles de réécriture du changement RemoveClass (Employee).

5 Travaux liés

Malgré son importance, l'évolution des ontologies reste encore peu traité. Les premières méthodologies proposées dans la littérature étaient celle de (Stojanovic *et al.*, 2002; Klein, 2004) qui présentent la base de la plupart des travaux actuels. Ainsi, (Hartung *et al.*, 2012) ont étudié l'évolution et

la différence entre deux versions d'une même ontologie. Le travail offre un outil *COnto-Diff* capable de détecter les différents changements élémentai res, par contre, il ne présente aucun traitement pour les incohérences.
(Luong, 2007; Khattak *et al.*, 2013) ont proposé des approches à postériori de résolution des incohérences. Ce type de démarche, à la différence du processus à priori, nécessite l'exécution des changements pour pouvoir juger l'altération de l'ontologie puis l'annuler en cas de problème. Ceci provoque une perte de temps et de ressource. Le travail de (Djedidi & Aufaure, 2009) a remédié ce problème en proposant une approche à priori basée sur le moteur d'inférence Pellet. En effet, Pellet offre différentes fonctionnalités mais souffre de certaines limites (Djedidi, 2009). (Dragoni & Ghidini, 2012) ont aussi traité l'impact de l'évolution des ontologies. Ils considèrent l'ontologie comme étant une hiérarchie de concepts en négligeant ainsi les relations conceptuelles et sémantiques qu'elle modélise.

L'utilisation des grammaires de graphes présente un champ favorable de formalisation et d'application des changements ontologiques. Son point fort est essentiellement sa capacité à définir un processus d'adaptation à priori permettant de vérifier grâce aux conditions d'application la validité de chaque type de changement et ses effets sur le reste du graphe.

6 Conclusion & perspectives

Nous avons présenté dans cet article la formalisation et l'application des changements ontologiques à l'aide des grammaires de graphes. Les grammaires de graphes permettent grâce aux règles de réécriture de définir un changement ontologique et de suivre son application pour garantir la cohérence de l'ontologie modifiée. Le choix de l'outil AGG a offert un cadre visuel permettant de définir les règles de réécriture d'une manière graphique simple. Il nous a permis aussi, grâce à son API Java d'automatiser le processus de transformation des ontologies sous forme de graphe et inversement. De nombreuses perspectives peuvent être dégagées à la suite de ce travail. En premier abord, il s'agit d'étendre l'étude des changements ontologiques complexes, à savoir fusionClasses, moveClasse, etc. Il serait également intéressant d'exploiter les changements ontologiques pour définir une approche de composition d'ontologies. Une étude comparative avec les graphes conceptuels (Chein & Mugnier, 2009) est envisagée sachant que ce formalisme possède aussi un potentiel important et offre une API Cogitant [4] facilitant son utilisation.

4. http ://cogitant.sourceforge.net/

Références

CHEIN M. & MUGNIER M.-L. (2009). *Graph-based knowledge representation : computational foundations of conceptual graphs*. Springer.

DJEDIDI R. (2009). *Approche d'évolution d'ontologie guidée par des patrons de gestion de changement.* PhD thesis, Université Paris Sud-Paris XI.

DJEDIDI R. & AUFAURE M.-A. (2009). Patrons de gestion de changements owl. In *Ingénierie des Connaissances*, p. 145–156.

DRAGONI M. & GHIDINI C. (2012). Evaluating the impact of ontology evolution patterns on the effectiveness of resources retrieval. In *2nd Joint Workshop on Knowledge Evolution and Ontology Dynamics EvoDyn 2012*.

EHRIG H. (1979). Introduction to the algebraic theory of graph grammars (a survey). In *Graph-Grammars and Their Application to Computer Science and Biology*, p. 1–69 : Springer.

EHRIG H., PFENDER M. & SCHNEIDER H. J. (1973). Graph-grammars : An algebraic approach. In *Switching and Automata Theory, 1973. SWAT'08. IEEE Conference*, p. 167–180.

ERMEL. C., RUDOLF. M. & TAENTZER G. (1999). The agg approach : Language and environment. In *Handbook of graph grammars and computing by graph transformation*, p. 551–603 : World Scientific Publishing Co., Inc.

GRUBER T. R. (1993). A translation approach to portable ontology specifications. *Knowledge acquisition*, **5**(2), 199–220.

HARTUNG M., GROSS A. & RAHM E. (2012). Conto-diff : Generation of complex evolution mappings for life science ontologies. *J Biomed Inform*.

KHATTAK A. M., LATIF K. & LEE S. (2013). Change management in evolving web ontologies. *Knowledge-Based Systems*.

KLEIN M. (2004). *Change Management for Distributed Ontologies*. PhD thesis, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands.

LÖWE M. (1993). Algebraic approach to single-pushout graph transformation. *Theoretical Computer Science*, **109**(1), 181–224.

LUONG P. H. (2007). *Gestion de l'évolution d'un Web sémantique d'entreprise*. PhD thesis, École Nationale Supérieure des Mines de Paris.

ROZENBERG G. (1999). *Handbook of graph grammars and computing by graph transformation*, volume 1. World Scientific.

STOJANOVIC N., STOJANOVIC L. & HANDSCHUH S. (2002). Evolution in the ontology-based knowledge management system. In *Proceedings of the European Conference on Information Systems-ECIS*.